Τηλεπικοινωνιακά Συστήματα

Διάλεξη 9η

Θωμάς Καμαλάκης

Χαροκόπειο Πανεπιστήμιο Αθηνών

Οκτώβριος 2020

Περιεχόμενα

Η διαμόρφωση QAM

- Στη διαμόρφωση PSK, τα σύμβολα τοποθετούνται σε έναν κύκλο.
- Η κυματομορφή είναι

$$Y(t) = \sum_k c_k p_1(t - kT_S) + \sum_k s_k p_2(t - kT_S)$$

$$c_k^2 + s_k^2 = 1$$

- Ωστόσο δεν είναι απαραίτητο να τοποθετήσουμε τα σύμβολα σε έναν κύκλο όπως στο PSK.
- Στην γενικότερη περίπτωση όπου δεν είναι απαραίτητο αυτό, μιλάμε για την διαμόρφωση QAM - Quadrature Amplitude Modulation.

Η διαμόρφωση QAM

- Στην πιο απλή περίπτωση τα c_k και τα s_k μπορούν να τοποθετηθούν πάνω σε ένα τετράγωνικο αστερισμό.
- Ας υποθέσουμε ότι έχουμε $M = 2^{2n}$ σύμβολα QAM.
- Τότε μπορούμε να τα τοποθετήσουμε σε ένα τετραγωνικό αστερισμό $2^n \times 2^n$.

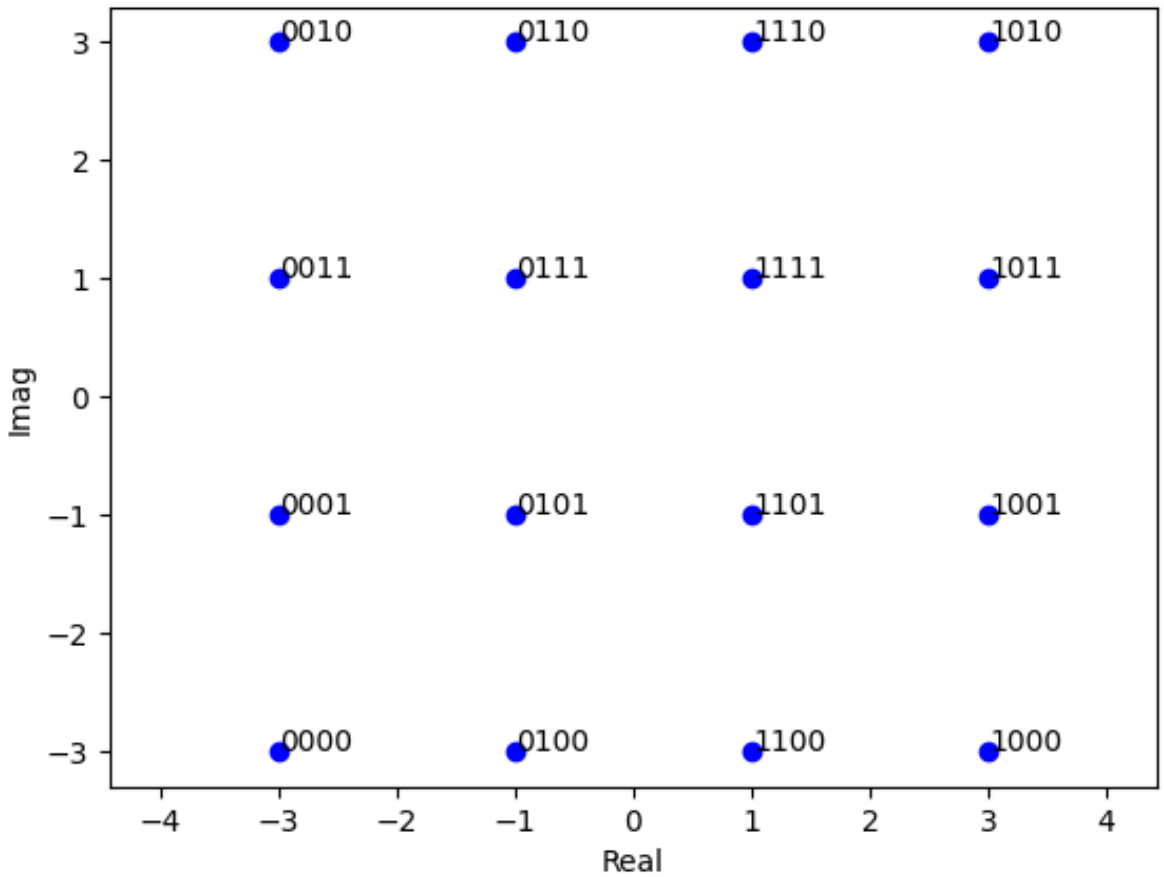
$$z_{pq} = \beta(2p - \sqrt{M} - 1) + j\beta(2q - \sqrt{M} - 1)$$

- Είναι δηλαδή σαν να έχουμε δύο οριζόντια μεταξύ τους PAM, κάθε ένα με $\sqrt{M} = 2^n$ σύμβολα.

Η διαμόρφωση QAM

```
class qam_constellation(constellation):

    def set_gray_bits( self, m ):
        ms = int(m / 2)
        g = gray_code( ms )
        self.bits = []
        self.bits_str = []
        self.map = {}
        self.m = m
        i = 0

        for p, cwp in enumerate(g):
            for q, cwq in enumerate(g):
                cw = cwp + cwq
                self.bits_str.append(cw)
                self.bits.append(
      str_to_bitsarray( cw ) )
                self.map[ cw ] = self.symbols[ i
      ]
                i += 1

    def __init__(self, M, beta = 1, title = None,
        SNRbdB = None):
        super().__init__(title = title)

        self.M = M
        self.m = np.log2(M).astype(int)
        self.SNRbdB = SNRbdB
        Ms = np.sqrt(M).astype(int)
        self.Ms = Ms
        i = 0
        symbols = np.zeros( M, dtype = complex )
        for p in range( Ms ):
            for q in range( Ms ):
                symbols [ i ] = beta * ( 2*p - Ms
      + 1 ) + 1j * beta * ( 2*q - Ms + 1 )
                i += 1

        self.set_symbols( symbols )
        self.set_gray_bits( self.m )

    def ser(self):
        SNRb = 10 ** ( self.SNRbdB / 10)
        q2 = 3 * SNRb * self.m / (self.M - 1)
        q = np.sqrt(q2)
        P = 2 * ( np.sqrt(self.M) - 1 ) / np.sqrt
      (self.M) * Qfunction(q)
        return 1 - (1-P) ** 2

    def ber(self):
        return self.ser() / np.log2(self.M)
```

Η διαμόρφωση QAM



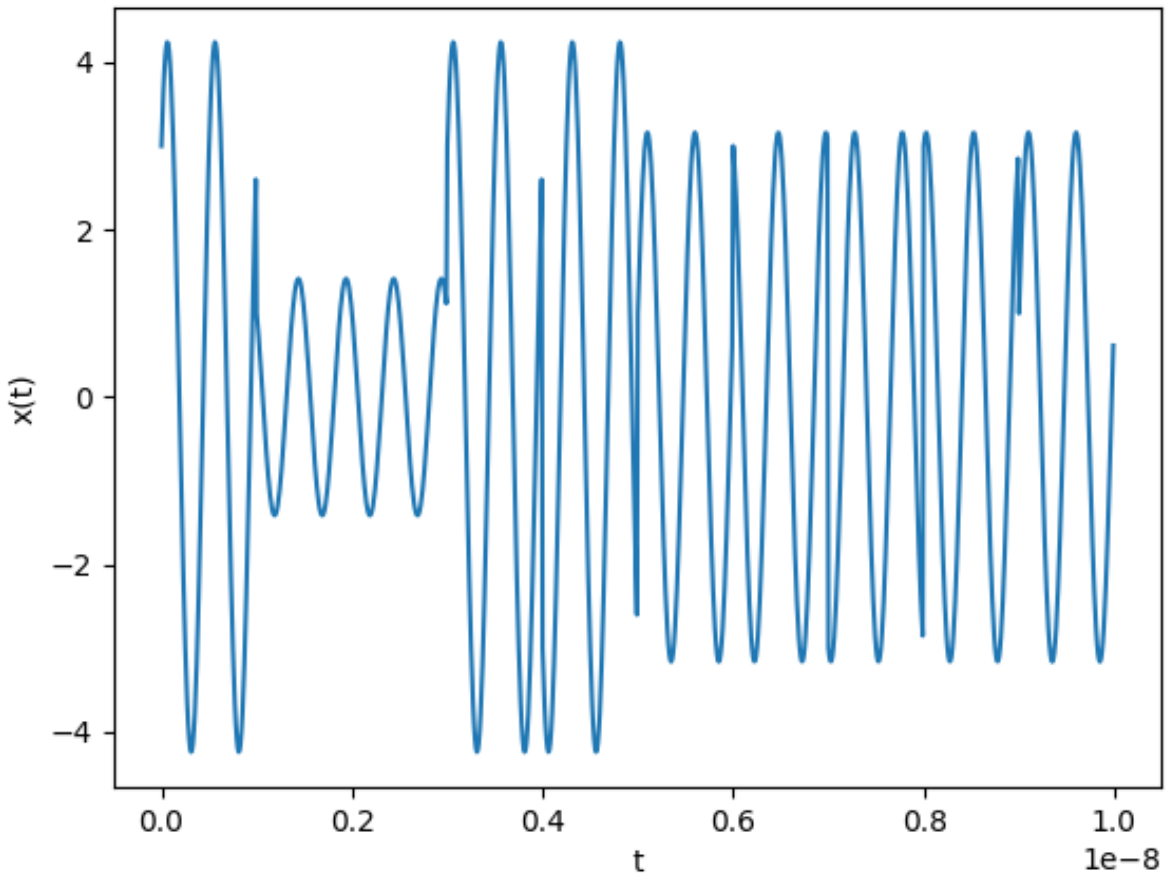
Η διαμόρφωση QAM, $M = 16$

- Όπως φαίνεται και στην προηγούμενη διαφάνεια, για $M = 2^{2n}$ χωρίζουμε τα $2n$ bit σε δύο ομάδες των n bits η κάθε μία.
- Σε κάθε ομάδα χρησιμοποιούμε τον κώδικα Gray μήκους n bits.
- Η πρώτη ομάδα αντιστοιχεί στο πραγματικό μέρος ενώ η δεύτερη το φανταστικό.
- Στην περίπτωση αυτή μπορούμε να κάνουμε την αποκωδικοποίηση σε δύο στάδια: μία για το πραγματικό μέρος και μία για το φανταστικό μέρος των συμβόλων.

Η διαμόρφωση QAM

```
class qam_constellation(constellation):

    def set_gray_bits( self, m ):
        ms = int(m / 2)
        g = gray_code( ms )
        self.bits = []
        self.bits_str = []
        self.map = {}
        self.m = m
        i = 0

        for p, cwp in enumerate(g):
            for q, cwq in enumerate(g):
                cw = cwp + cwq
                self.bits_str.append(cw)
                self.bits.append(
      str_to_bitsarray( cw ) )
                self.map[ cw ] = self.symbols[ i
      ]
                i += 1

    def __init__(self, M, beta = 1, title = None,
        SNRbdB = None):
        super().__init__(title = title)

        self.M = M
        self.m = np.log2(M).astype(int)
        self.SNRbdB = SNRbdB
        Ms = np.sqrt(M).astype(int)
        self.Ms = Ms
        i = 0
        symbols = np.zeros( M, dtype = complex )
        for p in range( Ms ):
            for q in range( Ms ):
                symbols [ i ] = beta * ( 2*p - Ms
      + 1 ) + 1j * beta * ( 2*q - Ms + 1 )
                i += 1

        self.set_symbols( symbols )
        self.set_gray_bits( self.m )

    def ser(self):
        SNRb = 10 ** ( self.SNRbdB / 10)
        q2 = 3 * SNRb * self.m / (self.M - 1)
        q = np.sqrt(q2)
        P = 2 * ( np.sqrt(self.M) - 1 ) / np.sqrt
      (self.M) * Qfunction(q)
        return 1 - (1-P) ** 2

    def ber(self):
        return self.ser() / np.log2(self.M)
```

Η διαμόρφωση QAM



Ο δέκτης QAM

- Όπως και στην περίπτωση του PSK στην διάρκεια του $k^{\text{οστού}}$ συμβόλου δηλαδή για $kT_S \leq t < (k+1)T_S$ ο δέκτης λαμβάνει την κυματομορφή:

$$y(t) = c_k p_1(t) + s_k p_2(t) + n(t)$$

- και υπολογίζει τις δύο συνιστώσες y_1 και y_2

$$y_1 =< y(t), p_1(t) >= c_k + n_1$$

$$y_2 =< y(t), p_2(t) >= s_k + n_2$$

- Τα y_1 και y_2 είναι ανεξάρτητα μεταξύ τους επειδή:
- τα σύμβολα c_k και s_k επιλέγονται ανεξάρτητα (δημιουργούνται από διαφορετικές ομάδες bits)
- τα n_1 και n_2 είναι ανεξάρτητα μεταξύ τους.

Ο δέκτης QAM

- Ο δέκτης στην συνέχεια μπορεί να θεωρήσει ότι έχει δύο ανεξάρτητα συστήματα $\sqrt{M}$-PAM ένα για το y_1 και ένα για το y_2.
- Για κάθε ένα σύστημα τα σύμβολα απέχουν μεταξύ τους 2β
- Οι συνιστώσες θορύβου n_i είναι μεταξύ τους ανεξάρτητες με μηδενική μέση τιμή και διακύμανση:

$$\sigma^2 = \frac{N_0}{2}$$

- Για κάθε PAM σύστημα i η πιθανότητα σφάλματος *συμβόλου* είναι:

$$P = P_{\text{e,i}} = 2\frac{\sqrt{M}-1}{\sqrt{M}} Q\left(\frac{\beta}{\sigma}\right)$$

Ο δέκτης QAM

- Η ενέργεια του σήματος QAM είναι

$$\mathcal{E}_k = \mathbb{E}\left\{c_k^2 + s_k^2\right\} = 2\beta^2 \frac{M-1}{3}$$

- Το πηλίκο σήμα προς θόρυβο γράφεται:

$$\text{SNR}_\text{S} = \frac{\mathcal{E}_k}{N_0} = 2\beta^2 \frac{M-1}{3N_0}$$

- Το όρισμα μέσα στην συνάρτηση Q γράφεται:

$$q^2 = \frac{\beta^2}{\sigma^2} = \frac{2\beta^2}{N_0} = \frac{3\text{SNR}_\text{S}}{M-1} = \frac{3\text{SNR}_\text{b} \log_2 M}{M-1}$$

Πιθανότητα σφάλματος

- Η πιθανότητα να έχουμε σωστή μετάδοση του συμβόλου σε κάθε υποσύστημα PAM είναι $1 - P_{\mathrm{e,i}}$.
- Για το QAM, η πιθανότητα να έχουμε σωστή μετάδοση είναι:

$$P_{\mathrm{c}} = (1 - P_{\mathrm{e1}})(1 - P_{\mathrm{e2}}) = (1 - P)^2$$

- όπου η πιθανότητα P είναι:

$$P = 2\frac{\sqrt{M} - 1}{\sqrt{M}} Q\left(\frac{3\mathrm{SNR_b}\log_2 M}{M - 1}\right)$$

- Η πιθανότητα σφάλματος συμβόλου θα είναι

$$P_{\mathrm{e}} = 1 - (1 - P)^2$$

- Η πιθανότητα σφάλματος bit θα είναι

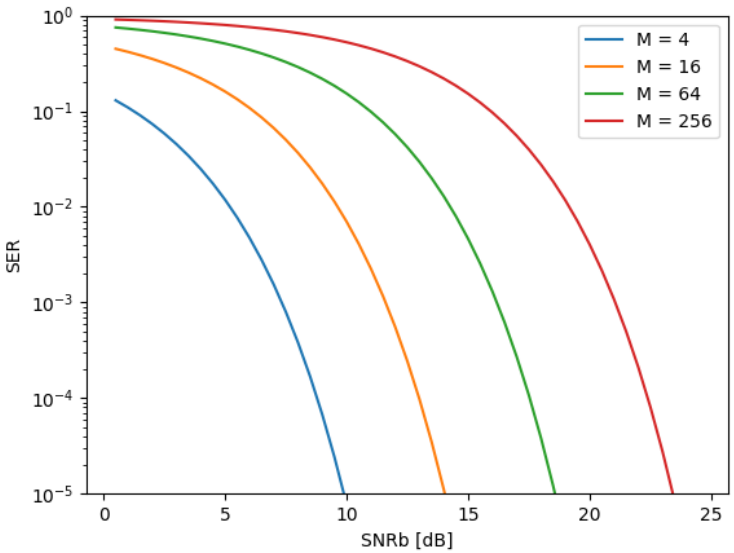
$$P_{\mathrm{b}} = \frac{P_{\mathrm{e}}}{\log_2 M}$$

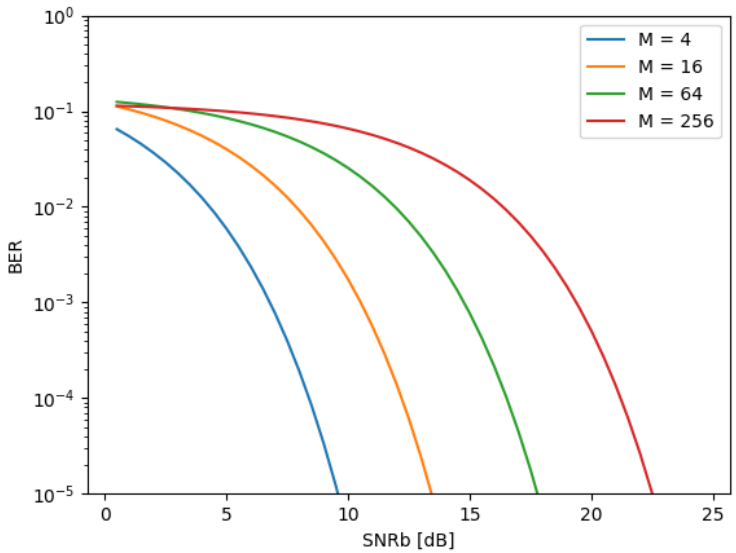
Πιθανότητα σφάλματος - qamservsM.py

```
from commlib import qam_constellation
import matplotlib.pyplot as plt
import numpy as np

SNRbdBs = np.arange(0.5, 25, 0.5)
n = np.arange(1,7,1)
Ms = np.array([4, 16, 64, 256])
Ms = Ms.astype(int)

Pest = np.zeros( [SNRbdBs.size, Ms.size] )
Pebt = np.zeros( [SNRbdBs.size, Ms.size] )

threshold = 1e-4

for i, SNRbdB in enumerate(SNRbdBs):
    for j, M in enumerate(Ms):
        c = qam_constellation(M = M, SNRbdB =
       SNRbdB)
        Pest[i,j] = c.ser()
        Pebt[i,j] = c.ber()
        print('M = %d, SNRbdB = %6.2f Pe = %e' %
       (M,SNRbdB, Pest[i,j]))
```

```
plt.close('all')

for j, M in enumerate(Ms):
    plt.figure(1)
    plt.semilogy( SNRbdBs, Pest[:,j], label = 'M
       = %d' % M)
    plt.figure(2)
    plt.semilogy( SNRbdBs, Pebt[:,j], label = 'M
       = %d' % M)

plt.figure(1)
plt.xlabel('SNRb [dB]')
plt.ylabel('SER')
plt.legend()
plt.ylim([1e-5, 1])

plt.figure(2)
plt.xlabel('SNRb [dB]')
plt.ylabel('BER')
plt.legend()
plt.ylim([1e-5, 1])
```

Πιθανότητα σφάλματος





Σύγκριση συστημάτων - compber.py

```
from commlib import pam_constellation, psk_constellation, qam_constellation
import matplotlib.pyplot as plt
import numpy as np

SNRbdBs = np.arange(4, 30, 0.5)
Ms = np.array([16, 64])
Ms = Ms.astype(int)

Pest = {}
Pebt = {}
for key in ['pam','psk','qam']:
    Pest[key] = np.zeros( [SNRbdBs.size, Ms.size] )
    Pebt[key] = np.zeros( [SNRbdBs.size, Ms.size] )

for i, SNRbdB in enumerate(SNRbdBs):
    for j, M in enumerate(Ms):
        cpam = pam_constellation(M = M, SNRbdB = SNRbdB)
        Pest['pam'][i,j] = cpam.ser()
        Pebt['pam'][i,j] = cpam.ber()
        cpsk = psk_constellation(M = M, SNRbdB = SNRbdB)
        Pest['psk'][i,j] = cpsk.ser()
        Pebt['psk'][i,j] = cpsk.ber()
        cqam = qam_constellation(M = M, SNRbdB = SNRbdB)
        Pest['qam'][i,j] = cqam.ser()
        Pebt['qam'][i,j] = cqam.ber()
        print('M = %d, SNRbdB = %6.2f' % (M, SNRbdB) )
```

Σύγκριση συστημάτων - compber.py

```
plt.close('all')

for j, M in enumerate(Ms):
    plt.figure(1)
    plt.semilogy( SNRbdBs, Pest['pam'][:,j], '—d', label = 'PAM M = %d' % M)
    plt.semilogy( SNRbdBs, Pest['psk'][:,j], '—o', label = 'PSK M = %d' % M)
    plt.semilogy( SNRbdBs, Pest['qam'][:,j], '—s', label = 'QAM M = %d' % M)
    plt.figure(2)
    plt.semilogy( SNRbdBs, Pebt['pam'][:,j], '—d', label = 'PAM M = %d' % M)
    plt.semilogy( SNRbdBs, Pebt['psk'][:,j], '—o', label = 'PSK M = %d' % M)
    plt.semilogy( SNRbdBs, Pebt['qam'][:,j], '—s', label = 'QAM M = %d' % M)
    plt.figure(2)

plt.figure(1)
plt.xlabel('SNRb [dB]')
plt.ylabel('SER')
plt.legend()
plt.ylim([1e—5, 1])

plt.figure(2)
plt.xlabel('SNRb [dB]')
plt.ylabel('BER')
plt.legend()
plt.ylim([1e—5, 1])
```

Σύγκριση συστημάτων (PAM/PSK/QAM)